**Title:** Demonstrate function template for sorting algorithm..

**Objectives:** 1) To learn and understand templates.

2) To demonstrate function template for selection sort.

**Problem Statement:** Write a function template selection Sort. Write a program that inputs, sorts and outputs an integer array and a float array.

**Outcomes:** 1) Students will be able to learn and understand working and use of function template.

2) Students will be able to demonstrate function template for selection sort.

**Hardware requirements:** Any CPU with Pentium Processor or similar, 256 MB RAM or more, 1 GB Hard Disk or more.

**Software requirements:** 64 bit Linux/Windows Operating System, G++ compiler

**Theory:**

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type. A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept. There is a single definition of each container, such as vector, but we can define many different kinds of vectors for example, vector <int> or vector <string>.

You can use templates to define functions as well as classes, let us see how do they work:

Function Template: The general form of a template function definition is shown here:

template <typename type> ret-type func-name(parameter list)

{

// body of function

}

Here, type is a placeholder name for a data type used by the function. This name can be used within the function definition.

A function template behaves like a function except that the template can have arguments of many different types (see example). In other words, a function template represents a family of functions. The format for declaring function templates with type parameters is:

template <class identifier> function_declaration;

template <typename identifier> function_declaration;

Both expressions have the same meaning and behave in exactly the same way. The latter form was introduced to avoid confusion, since a type parameter need not be a class. (it can also be a basic type such as int or double.)

For example, the C++ Standard Library contains the function template max(x, y)which returns the larger of x and y. That function template could be defined like this:

template <typename T>

inline T max(T a, T b) {

return a > b ? a : b;

}

This single function definition works with many data types. The usage of a function template saves space in the source code file in addition to limiting changes to one function description and making the code easier to read.

A template does not produce smaller object code, though, compared to writing separate functions for all the different data types used in a specific program. For example, if a program uses both an int and a double version of the max() function template shown above, the compiler will create an object code version of max() that operates on int arguments and another object code version that operates on double arguments. The compiler output will be identical to what would have been produced if the source code had contained two separate non-templated versions of max(), one written to handle int and one written to handle double.

**Selection sort** is a simple sorting algorithm. This sorting algorithm is a in-place comparison based algorithm in which the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list.

Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues moving unsorted array boundary by one element to the right.

This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n are no. of items.

Step 1 − Set MIN to location 0

Step 2 − Search the minimum element in the list

Step 3 − Swap with value at location MIN

Step 4 − Increment MIN to point to next element

Step 5 − Repeat until list is sorted

**Algorithm:**

1. Define a function template with name input() and two arguments, one argument for array to sort and other for size of array
2. Accept size number of elements from user for sorting in input()
3. Define other function template display() with two arguments, one pointer of array to sort and other number of elements to sort
4. Print sorted elements stored in array
5. Define another function template sort() to implement selection sort for generic datatype.
6. Declare a variable min to hold index position of smallest element in array to sort.
7. Define for loop with loop variable " i " from 0 to size, incremented by one.
8. Assign first index position to min variable i.e. 0.
9. Define second for loop with variable " j " starting from "i + 1" to size, incremented by one.

10. In inner for loop, check if element at index position “ j ” is smaller than element at index position “ min”.
11. If condition is true, assign min as j else continue
12. In outer for loop, swap element at index position “ i ” with element at index position at “ min ”.
13. In main() method, declare variable size to accept size of array.
14. Input number of elements to sort and declare an array of given size.
15. Use int as datatype of array
16. Call input() function and pass array name and size as argument
17. Call sort() function and pass array name and size as argument to sort
18. Call display() function and pass array name and size as argument to display sorted array.
19. Repeat steps from 15 to 19 with datatype float
20. End program.

**Flowchart:**

**Draw proper flowchart and get it corrected from faculty.**

**Test Cases:**

**Test Case 1:** (All valid inputs are given and desired output is shown)

How many elements you want to sort : 5

Enter 5 elements in array : 10 32 2 -32 11

Elements before sorting : 10 32 2 -32 11

Elements after sorting : -32 2 10 11 32

Enter 5 elements in array : 1.30 3.2 4.2 -4.32 1.1

Elements before sorting : 1.30 3.2 4.2 -4.32 1.1

Elements after sorting : -4.32 1.1 1.30 3.2 4.2

**Test Case 2:** (When size is given as negative value)

How many elements you want to sort : -4

Array size can not be negative. Initializing size to 1

Enter 1 elements in array : 10

Elements before sorting : 10

Elements after sorting : 10

Enter 1 elements in array : 1.1

Elements before sorting : 1.1

Elements after sorting : 1.1

**Conclusion:** Hence, we demonstrated use of function template for selection sort

# PROGRAM

```
#include<iostream>
using namespace std;
template <class Z>
class templates
{
      Z a[100];
      int num;
public:
      void accept()
      {
            int i;
            cout<<"Enter the number of elements in your array"<<endl;
            cin>>num;
            cout<<"Enter the elements of the array"<<endl;
            for(i=0;i<num;i++)
                  cin>>a[i];
      }
      void selsort()
      {
            int i,j,min;

            for(int i=0;i<num;i++)
{
    min=i;
for(int j=i+1;j<num;j++)
{
    if(a[j]<a[min])
    min=j;
```

```
}
swap(a[i],a[min]);

}
            cout<<"Sorted array:"<<endl;
            for(i=0;i<num;i++)
                  cout<<a[i]<<"\n";
      }
};


int main()
{
      int ch;
      templates<int> obj;
      templates<float> obj1;
      cout<<"1- Selection sort for integer array\n2- Selection sort for float
array\nEnter choice"<<endl;
      cin>>ch;
      switch(ch)
      {
      case 1:
            obj.accept();
            obj.selsort();
            break;
      case 2:
            obj1.accept();
            obj1.selsort();
            break;
      default:
            cout<<"Wrong choice entered"<<endl;
      }
      return 0;
}
```